SIMPLE LINEAR REGRESSION

In machine learning, linear regression is a commonly used technique for modeling the relationship between a dependent variable and one or more independent variables, also known as features or predictors.

Why Linear Regression?

Linear regression in machine learning is a popular and widely used technique because it is relatively **simple and interpretable**, and it can often achieve good performance on a wide range of problems. Also whatever we learn in linear regression, we can implement in other machine learning algorithms.

Linear regression is a supervised learning algorithm, which means that it requires labeled data to train the model. There are several types of linear regression that can be used depending on the specific problem and data at hand. Some of the most common types of linear regression include:

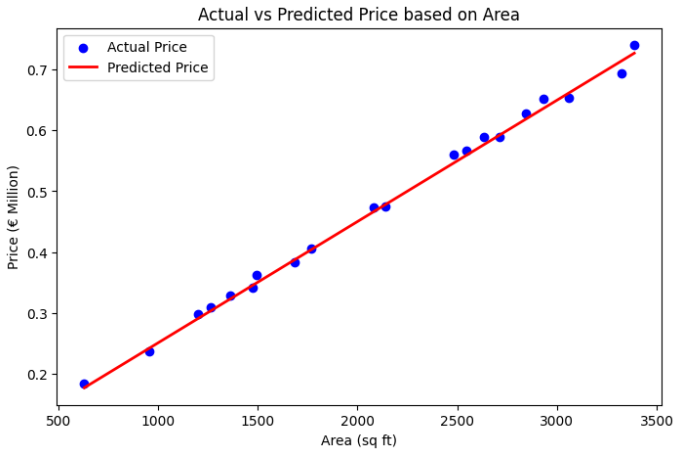
1. Simple linear regression: This is the most basic type of linear regression, where there is only one independent variable and one dependent variable. The goal is to find a linear relationship between the two variables.
2. Multiple linear regression: This type of linear regression involves more than one independent variable, and the goal is to find a linear relationship between the independent variables and the dependent variable.
3. Polynomial regression: This type of linear regression is used when the relationship between the independent variable and the dependent variable is not linear but can be approximated by a polynomial function.

Regression can be applied to various types of data. The **input features** can be different **types of datasets,** including numerical, categorical, image, text, video, etc. The goal of regression is to predict a **continuous numerical target variable** or parameter based on the input features. It should be noticed that deep learning methods are the most widely used approaches for working with audio, image, and video data on ML. This is because deep learning algorithms are able to extract complex patterns from data, which is essential for these tasks.

Let's explore how regression can be applied to different types of data with examples:

1. **Numerical Data:**

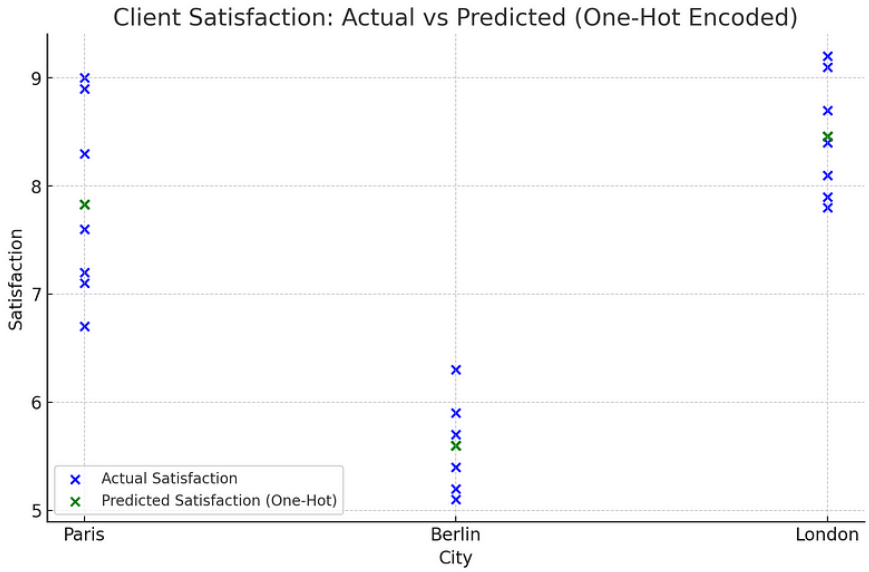
- Example: Predicting house prices based on features like the number of bedrooms, square footage, and distance to the nearest amenities.
- Features: Number of bedrooms (numerical), Square footage (numerical), Distance to amenities (numerical)
- Target: House price (continuous numeric)



House price prediction by regression and in terms of area

2. Categorical Data with Numeric Values (Ordinal):

- Example: Predicting customer satisfaction scores based on categorical feedback.
- Features: Feedback city (categorical: Berlin, Paris, London)
- Target: Customer satisfaction score (continuous numeric)



Client satisfaction prediction: the predicted score are 5.6, 7.2 and 8.4 for Berlin, Paris and London.

3. Text Data (Natural Language Processing Regression):

- Example: Predicting the sentiment score of customer reviews.
- Features: Textual content (text data), Keywords or sentiment scores extracted from text
- Target: Sentiment score (continuous numeric)

Review	Sentiment_Score
Terrible product, it broke within days.	1.5
I love the quality, very pleased with the purchase.	9.2
Very poor customer support, I'm extremely unhappy.	2.0
The experience was average, nothing special.	5.0
Fast shipping and great customer service!	8.8
Product arrived damaged, highly disappointed.	2.5
I will definitely buy again, the quality is amazing.	9.4

Table shows customer review sentences with their corresponding sentiment scores. The sentence used for prediction was: **"The product quality is excellent, but the delivery was slow."** The **predicted sentiment score** for this sentence is approximately **7.93**.

4. Image Data:

- Example: Estimating the age of a person in an image.
- Features: Pixel values from the image, Features extracted from pre-trained convolutional neural networks (CNNs)
- Target: Age of the person (continuous numeric)

*** We can use traditional regression methods (like Decision Trees, or Support Vector Regression) to estimate a person's age from an image, but the performance will likely not match the accuracy of deep learning models like Convolutional Neural Networks (CNNs), which are well-suited for capturing the complex features in image data. However, if you prefer using classic regression techniques, you can first extract meaningful features such as texture, edges, and histograms from the images and then apply regression methods to those features.

5. Audio Data:

- Example: Predicting the duration of a sound event.
- Features: Spectrogram or other audio representations, Features extracted from audio signals

- Target: Duration of the sound event (continuous numeric)

** While **traditional regression methods** can be used for audio tasks like **voice duration estimation,** deep learning methods (especially models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), including LSTMs) often yield significantly better results. This is because deep learning models can automatically learn complex, hierarchical patterns from raw or minimally processed audio data, whereas traditional regression models rely on manually extracted features, which may miss subtle but important patterns. In **traditional regression methods, y**ou need to extract features like **MFCC**, **zero-crossing rate (ZCR)**, **pitch**, **spectral features**, etc., which can describe the audio signal in a more structured way. After feature extraction, you can apply regression methods like: **Linear Regression, Random Forest Regressor, Support Vector Regression (SVR).**

6. Time Series Data:

- Example: Predicting future stock prices.
- Features: Historical stock prices (numeric), Economic indicators (numeric)
- Target: Future stock price (continuous numeric)

** **Traditional regression methods** are useful are generally not ideal for time series prediction. For **better performance**, consider using time-series models like ARIMA, GARCH, or deep learning models like LSTM, which are specifically designed for handling sequential, time-dependent data.

7. Spatial Data:

- Example: Predicting property values based on geographic features.
- Features: Geographic coordinates (numerical), Area-specific characteristics (numeric)
- Target: Property value (continuous numeric)

8. Video Data:

- Example: Predicting the viewer engagement duration of an online advertisement video.
- Features: Frame-level features: Extracted visual elements, colors, and composition from each frame of the video (**Example**: Videos with bright, vibrant colors might catch a viewer's attention more than dull, dark videos). Audio features: Audio intensity over time, sentiment analysis of background music or voiceover (**Example**: Videos with dynamic, upbeat background music may keep viewers engaged longer than videos with monotonous or low-energy music). Content-specific features: Scene changes, object recognition (e.g., presence of a product), facial expressions of people in the video.
- Target: Duration of viewer engagement with the advertisement (continuous numeric).

** We can use traditional regression methods (such as **Decision Trees**, **Random Forest**, or **Support Vector Regression**) to predict **viewer engagement duration** from the extracted features of a video. However, the performance might not be as high as more specialized deep learning models like **Convolutional Neural Networks (CNNs)** or **Recurrent Neural Networks (RNNs)**, which are more suited for handling complex temporal and visual patterns in video data. If you prefer to stick with traditional regression methods, you will need to **manually extract meaningful features** from the video (similar to extracting texture, edges, and histograms in the age prediction example) and then apply regression to these features.

9. Graph Data:

- Example: Predicting the traffic flow on a road network based on various graph features.
- Features: Nodes representing intersections or locations, Edges representing road segments, Traffic-related features (e.g., historical traffic flow, time of day)
- Target: Traffic flow (continuous numeric)

** Traditional regression methods, such as linear regression , are generally not well-suited for graph data because they assume independence between data points. In graph data, nodes and edges represent relationships, and these relationships violate the assumption of independence. This interconnected structure requires specialized methods to account for the dependencies between nodes.

To handle graph data more effectively, you can consider the following approaches:

1. **Graph-based regression methods**:

- **Graph Convolutional Networks (GCNs)**:
- **Graph-based kernel methods**:
- **Graph regularized regression**:

2. **Node embeddings**: You can use techniques like node2vec, DeepWalk, or GCNs to convert graph nodes into fixed-size vectors (embeddings), which can then be used as input features in traditional regression models.

Let's Understand the Simple Linear Regression in depth.

Simple Linear Regression

Simple linear regression is a statistical method used to model the relationship between two variables by fitting a linear equation to the data. One variable (usually denoted as "x") is considered as the predictor or independent variable, while the other variable (usually denoted as "y") is the response or dependent variable.

The equation for simple linear regression is:

$y = \beta 0 + \beta 1 * x + \varepsilon$

Where,
y — dependent variable
x — independent variable
β0 — intercept
β1 — coefficient of x
ε — error term

In simple linear regression, we try to find the best-fit line that explains the relationship between x and y. This line is called the regression line.

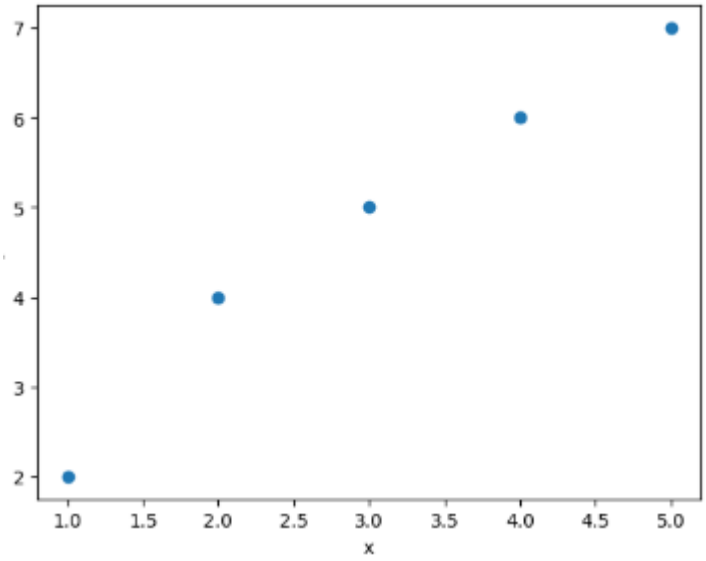
Code Example

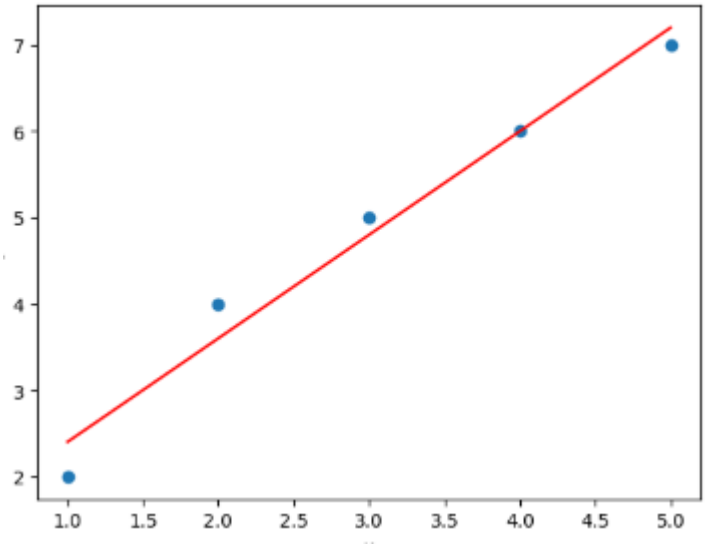
Here's an example of simple linear regression in Python:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# generate some random data for x and y
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 6, 7])

# visualize the data using scatter plot
plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()

# create a linear regression model and fit the data
model = LinearRegression()
model.fit(x.reshape(-1, 1), y)

# predict the values of y using the model
y_pred = model.predict(x.reshape(-1, 1))

# visualize the regression line
plt.scatter(x, y)
plt.plot(x, y_pred, color='red')
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.show()

# print the coefficients of the linear regression model
print('Intercept:', model.intercept_)
print('Coefficient:', model.coef_)
```

In this example, we generated some random data for x and y and visualized the data using a scatter plot. We then created a linear regression model using the LinearRegression class from scikit-learn, and fitted the data to the model. We then used the model to predict the values of y for the given values of x. Finally, we visualized the regression line and printed the coefficients of the linear regression model.





The output of this code will be:

Intercept: 1.5
Coefficient: [1.1]

This means that the intercept of the regression line is 1.5, and the coefficient of x is 1.1. Therefore, the equation of the regression line is:

y = 1.5 + 1.1*x

This equation can be used to predict the value of y for any given value of x.

Intuition

In the example code I provided, we were trying to model the relationship between x and y, where x was the independent variable and y was the dependent variable. The scatter plot of the data showed that there was a positive correlation between x and y, which means that as x increases, y also increases.

The linear regression model we created used the data to find the best-fit line that could explain this relationship. The line was found by minimizing the sum of the squared distances between the actual data points and the predicted values on the line. This line can then be used to predict the values of y for any given value of x.

In the output of the code, we can see that the intercept of the line is 1.5 and the coefficient of x is 1.1. This means that if x=0, then y=1.5, and for every unit increase in x, y increases by 1.1.

Overall, simple linear regression is a valuable tool for understanding the relationship between two variables and can be used to make predictions about the dependent variable based on the independent variable.

How to find m(slop) and b(intercept)

The formula for the slope (m) and intercept (b) in simple linear regression can be derived as follows:

Let (x1, y1), (x2, y2), ..., (xn, yn) be the n data points. We want to find the values of m and b that minimize the sum of the squared differences between the actual y values and the predicted y values on the line. This can be written as:

$\min \Sigma(yi - (mx + b))^2$

To find the values of m and b that minimize this expression, we can take partial derivatives with respect to m and b and set them equal to zero.

Taking the partial derivative with respect to m, we get:

$\partial/\partial m \, \Sigma(yi - (mx + b))^2 = -2\Sigma xi(yi - (mx + b))$

Setting this equal to 0 and solving for m, we get:

$\Sigma xiyi - nxb = m\Sigma xi^2$

Dividing both sides by Σxi^2, we get:

$m = (\Sigma xiyi - nxb) / \Sigma xi^2$

Taking the partial derivative with respect to b, we get:

$\partial/\partial b \, \Sigma(yi - (mx + b))^2 = -2\Sigma(yi - (mx + b))$

Setting this equal to 0 and solving for b, we get:

$\Sigma y_i - m\Sigma x_i = nb$

Dividing both sides by n, we get:

$b = \bar{y} - m\bar{x}$

where $\bar{y}$ and $\bar{x}$ are the means of the y and x values, respectively. This gives us the formula for the slope (m) and intercept (b) in terms of the sample means and sums of products and squares. Once we have calculated m and b using these formulas, we can use them to create the regression equation:

$y = mx + b$

This equation represents the line of best fit that describes the relationship between x and y in the data.

Diving into the code

Problem Statement : Salary V/S Experience

We'll look at a dataset with two variables: salary (the dependent variable) and experience (the Independent variable). This problem's objectives are as follows:

We want to see if there is any relationship between these two variables and how changing the independent variable affects the dependent variable. In this section, we will build a Simple Linear Regression model to determine which line best represents the relationship between these two variables.

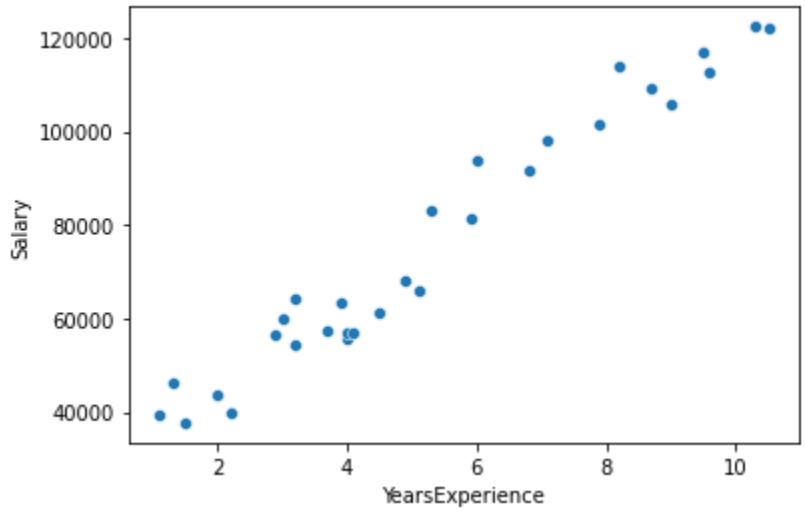
To use Python to create the Simple Linear Regression model in machine learning, follow the steps below:

1. Data Pre-Processing

```
# import libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

# mount drive
from google.colab import drive
drive.mount('/content/drive')
```

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

sns.scatterplot(x=df.YearsExperience, y = df.Salary)



Inference

Looking at the graph, we can see that salary(y) and years of experience(x) have a linear connection and are correlated. The graph also shows a rise in salary as the number of years of experience increases. Let's use the information provided to further predict the salary depending on the year of experience.

Following that, we must extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Here's the code:

```
x = df.iloc[:,:-1]     # independent variable (yoe)
y = df.iloc[:,1:]      # dependent variable (salary)

print(x.shape)
>> (30, 1)
```

Since we have extracted the independent variable (x) and the dependent variable (y), we will divide both variables into training and testing sets. We have 30 observations, so we will use 20 for the training set and 10 for the test set. We are separating our dataset so that we may train our model on one dataset and then test it on another.

Splitting the dataset into training and testing set

```
# splitting the dataset into training and testing set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=1/3, random_state=0)
print(x_train.shape)
print(y_test.shape)

>> (20, 1)
>> (10, 1)
```

We will not use Feature Scaling for simple linear regression. Now that our dataset is ready for analysis, we will begin creating a Simple Linear Regression model for the stated problem.

2. Fitting the Simple Linear Regression to the Training Set

Now the next step is to find the best fit line where the difference between the predicted values and actual values is minimum. To accomplish so, we will import the **LinearRegression class** from the linear model package from **sci-kit learn**. After importing the class, we will build a regressor class object.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

regressor.fit(x_train,y_train)

# getting model coefficient
regressor.coef_

>> array([[9345.94244312]])
```

We used the **fit() function** to fit our Simple Linear Regression object to the training set. We gave the x_train and y_train variables to the fit() method, which are our training datasets for the independent and dependent variables. We fitted our regressor object to the training set so that the model may readily learn the correlations between the predictor and target variables.

3. Prediction of Test Set result

As a result, our model is now prepared to predict the results for the fresh observations. In this stage, we'll give the model a test dataset (new observations) to see if it can predict the intended result.

A prediction vector called **y_pred** and **x_pred**, containing predictions for the test dataset and the training set, respectively, will be created.

prediction of test and training set

x_pred = regressor.predict(x_train)
y_pred = regressor.predict(x_test)

Two variables, y_pred and x_pred, are generated that include salary predictions for the training and test sets, respectively.

Quick Recap:

Till now, we have previously saved salary in y-variable and year of experience in x- variable, which was further split into training data and testing set.

We trained datasets by supplying x_train and y_train values to the model selection train_test_split() method. After successfully training the model, we predicted the testing and training set values, where we passed independent variables (x = year of experience) training and testing data to predict the salary for the training and testing set.

4. Visualizing the Training Set results

The training set result can be visualized graphically.

plt.scatter(x_train,y_train,color='blue')
plt.plot(x_train, x_pred, color= 'red')

plt.title("Salary vs Experience (Training Data)")
plt.xlabel('Years of Expereince')
plt.ylabel('Salary')
plt.show()

The **real values observations** are shown in **blue dots** in the graphic above, while the **predicted values** are covered by the **red regression line**. A correlation exists between the dependent and independent variables, as shown by the regression line.

The line's good fit can be seen by calculating the difference between actual and anticipated values. However, as seen in the accompanying graphic, the majority of the observations are close to the regression line, indicating that our model is suitable for the training set.

5. Visualizing the testing set

```
plt.scatter(x_test,y_test,color='red')
plt.plot(x_train, x_pred, color= 'green')

plt.title("Salary vs Experience (Training Data)")
plt.xlabel('Years of Expereince')
plt.ylabel('Salary')
plt.show()
```

The **red color** in the plot above represents **observations**, and the **green regression** line represents **prediction**. As we can see, the majority of the observations are close to the regression line, thus we can conclude that our Simple Linear Regression is a good model capable of making accurate predictions.